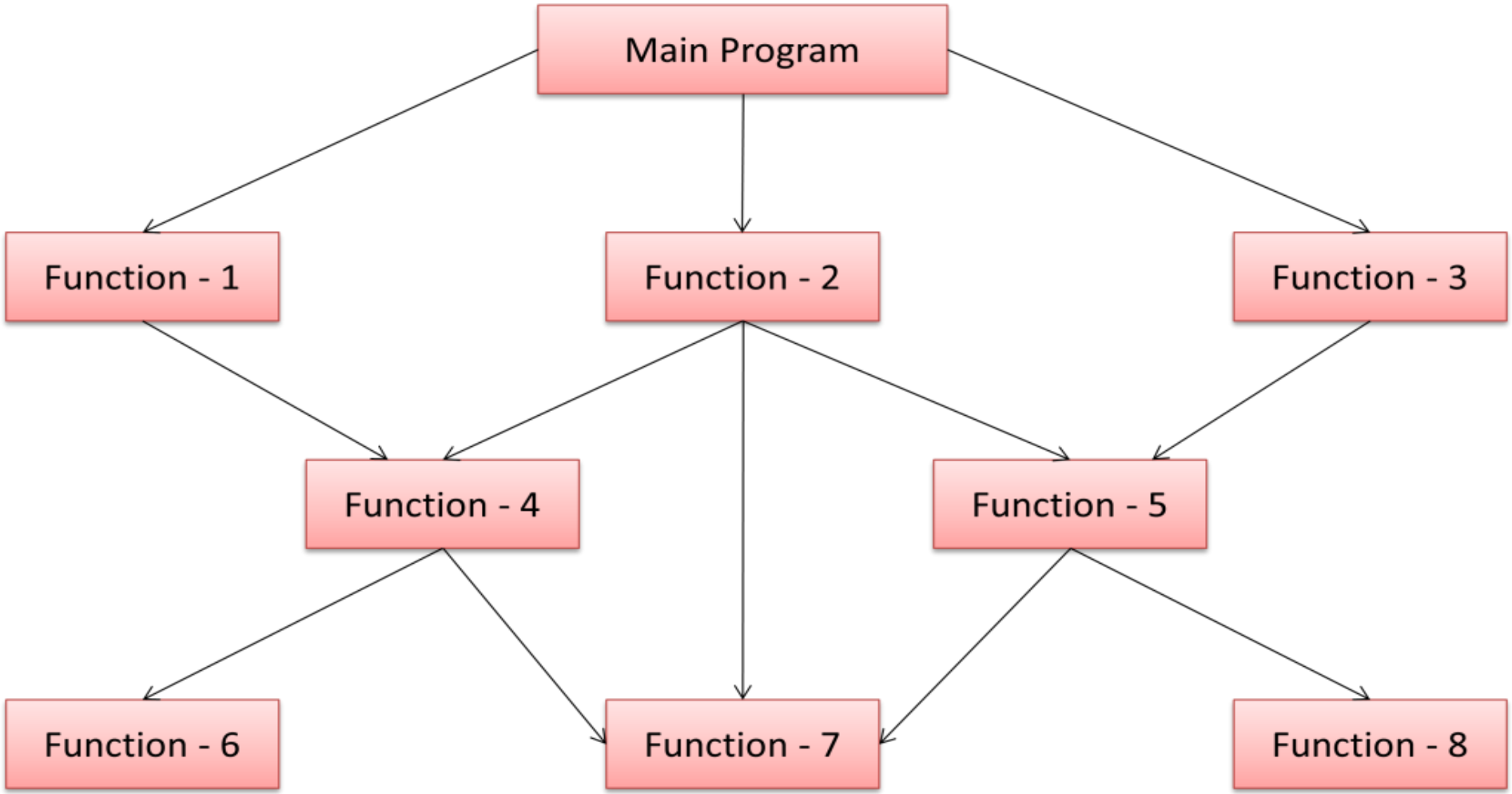
Unit –1: Principles of Object Oriented Programming

Overview of Structure Programming language

- A structure language is a type of computer programming language that specifies a series of well- structured steps and procedures within its programming context to compose a program.

- It contains a systematic order of statements, functions and commands to complete a computational task or program.



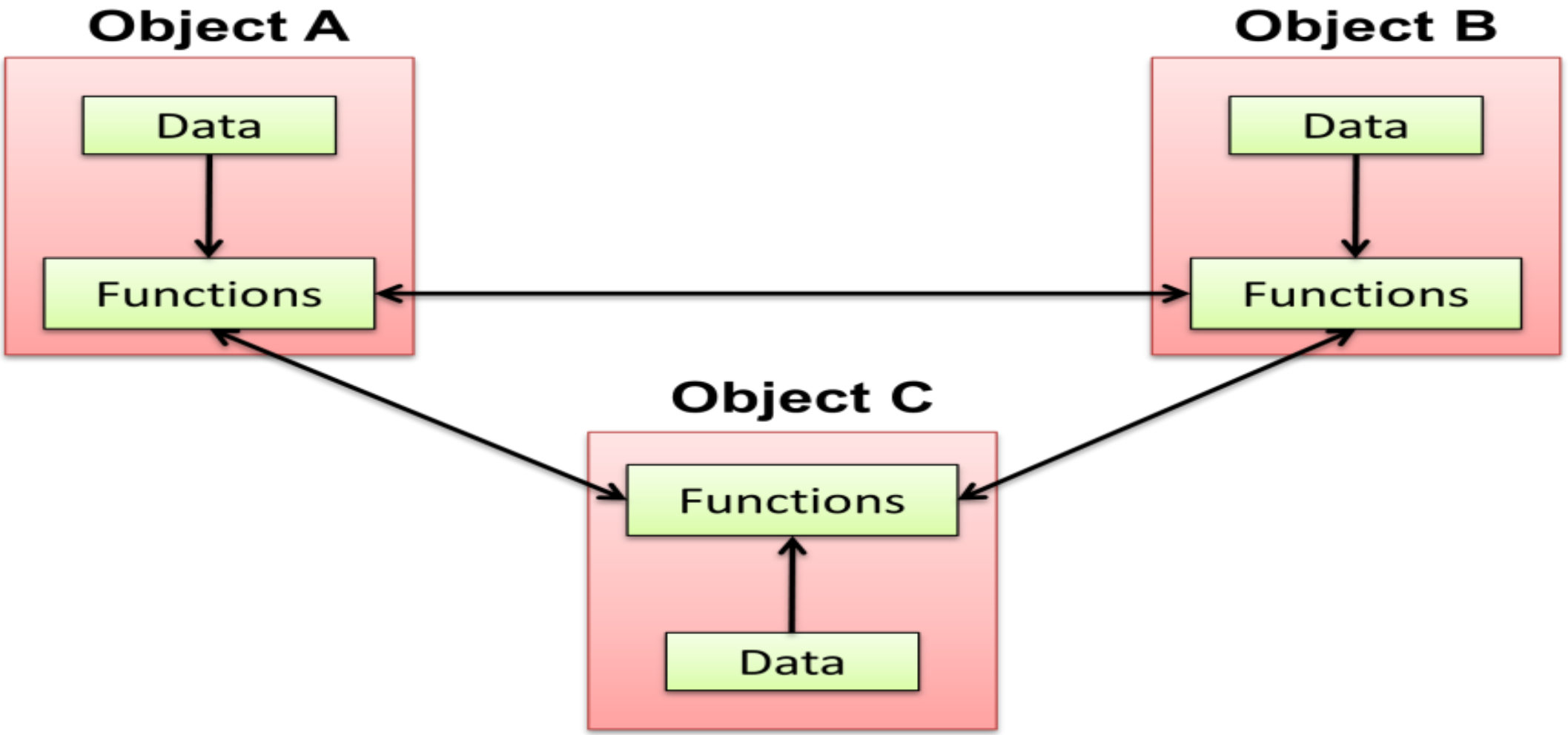
Structure of the procedure oriented programs

Disadvantage of POP:

- To develop a large program, it is very difficult to identify what data is used by which function.
- It does not model (solve) real world problem very well.

The object-oriented Approach:

- Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data.
- OOP treats data as a critical element in the program development and does not allow it to flow freely around the system.
- It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside function.
- OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.



Organization of data and function in OOP

Basic Concept of object-oriented Programming:

1) Object :

- Object can be defind as real world entities
- e.g. Person,Cycle,Chair, Pen, Window in computer software. Objects can be either

  Physical Object :Person, Table

  Conceptual Object: vehicle, bird

  Abstract Object: God
- An object is represented as its property (or attributes) and operations performed on it.

Object: Window	
Properties: size type position	**Operations:** change_size( ) move( ) minimize( ) maximize( ) close( )

```
#include <iostream>
using namespace std;
class Student {
        public:
                int id;               //data member (also instance variable)
                string name;  //data member(also instance variable)
        };
int main()
{
        Student s1;             //creating an object of Student
        s1.id = 201;
        s1.name = "Sonoo Jaiswal";
        cout<<s1.id<<endl;
        cout<<s1.name<<endl;
        return 0;
}
```

2) Classes:

- A class in C++ is a user defined type or data structure declared with keyword class that has data and functions as its members.
- A class definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces.

E.g.

```
class Box
{
public:
double length;          // Length of a box
double breadth;         // Breadth of a box double
height;                 // Height of a box
};
```

3) Abstraction:

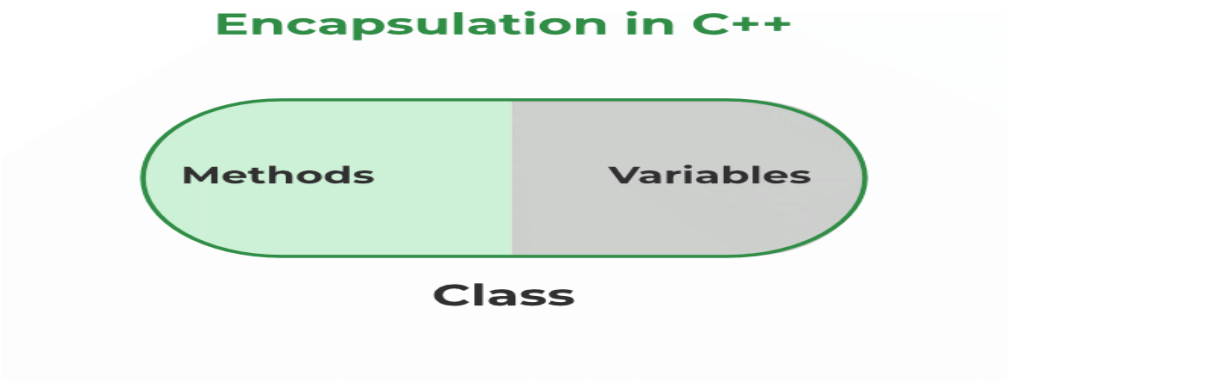
- Abstraction means keeping necessary while hiding unnecessary.
- Data abstraction is define as providing only essential information to the outside world and hiding their background details.

For example,

- when a person is working as employee in an organization, then in payroll system of organization, the salary is essential property, while weight is not essential.
- On other side if a person is a member of sports club then the weight is essential property, while salary is not essential.

4) Encapsulation:

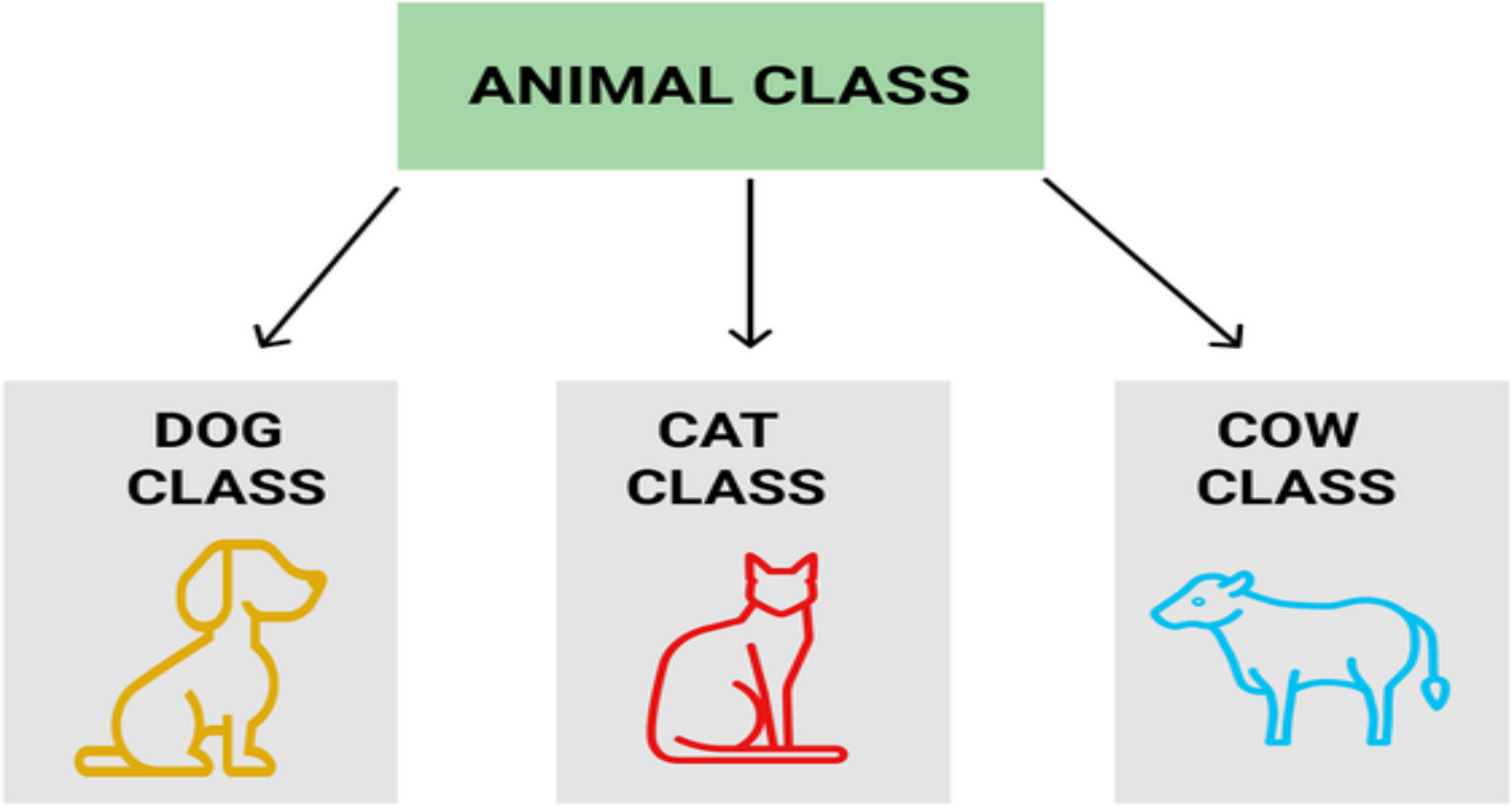
- Putting data and functions together in a single unit is known as encapsulation.
- It serves two purposes.
- It puts data near to the functions operating on it.
- It separates the external aspects of the object from internal implementation.
- In C++, encapsulation can be implemented using **classes** and access modifiers.

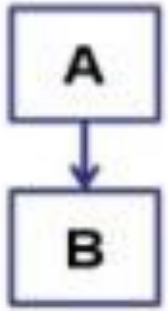


5) Inheritance:

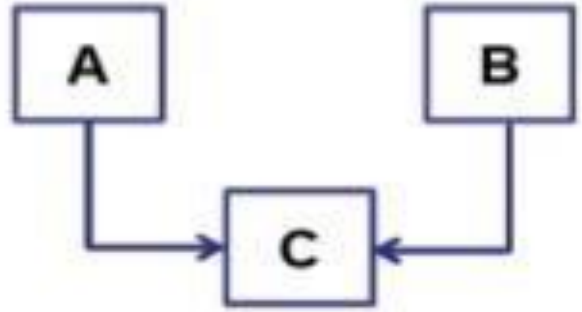
- The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.
- **Sub Class:** The class that inherits properties from another class is called Sub class or Derived Class.
- **Super Class:** The class whose properties are inherited by a sub-class is called Base Class or Superclass.
- **Reusability:** Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Example: Dog, Cat, Cow can be Derived Class of Animal Base Class.

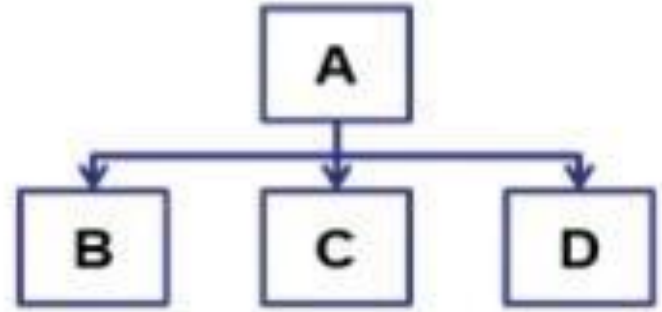




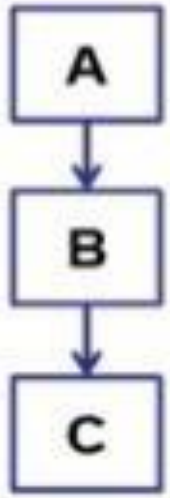
Single Inheritance



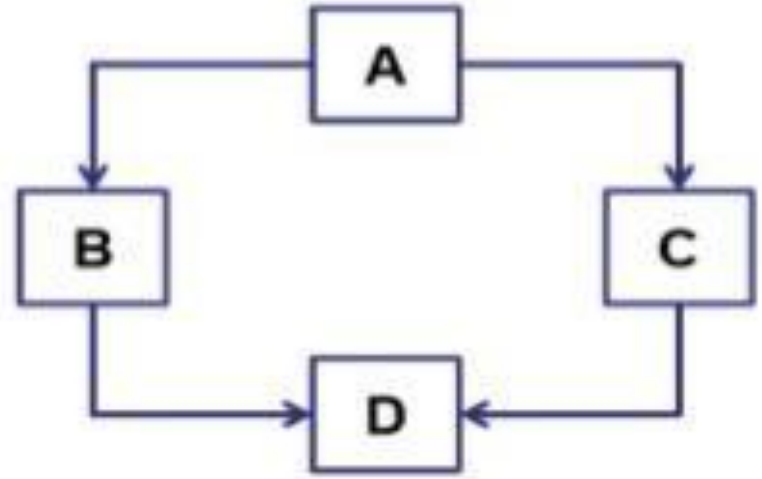
Multiple Inheritance



Hierarchical Inheritance



Multilevel Inheritance



Hybrid Inheritance

6) Polymorphism:

- Polymorphism means **"many forms",** and it occurs when we have many classes that are related to each other by inheritance.
- Like we specified in the previous chapter; Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.
- **For example**, think of a base class called Animal that has a method called animalSound(). Derived classes of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

- Polymorphism is that in which we can perform a task in multiple forms or ways. It is applied to the functions or methods.
- Polymorphism allows the object to decide which form of the function to implement at compile-time as well as run-time.
- **Types of Polymorphism are:**

1. Compile-time polymorphism (Method overloading)
2. Run-time polymorphism/Dynamic Binding(Method Overriding)

7) Message Passing:

- An object oriented program consists of a set of objects that communicate with each other.
- Steps for Message Passing:

1. Creating classes that define objects and their behavior.
2. Creating objects from class definition,
3. Establishing communication among objects.

- Objects communicate with one another by sending and receiving information.

Advantages of Object Oriented Programming

- Data is given primary importance which allows real world entities to be modeled easily.
- Object correspond to its counterpart in real world system i.e. entities which makes direct
- correspondence between problem and program. This enables easy understanding of the complex system.
- Data and functions are closed in a single unit(Class) which make data secured and protects from accidental changes or misuse.
- Once object is implemented, it can be used in many systems without developing every time from scratch which saves the time and make program more reliable.

Usage of Object Oriented Programming languages

- Real-time systems.
- Simulation and modelling.
- Object-oriented databases.
- Hypertext, hypermedia and expert text.
- AI and expert systems.
- Neural networks and parallel programming.
- Decision support and office automation systems.
- CIM/CAM/CAD systems.

Object Oriented Programming languages classification

1. Object Based Languages

- Object based languages supports the usage of object and encapsulation.
- They does not support inheritance or, polymorphism or, both.
- Object based languages does not supports built-in objects.
- Javascript, VB are the examples of object bases languages.

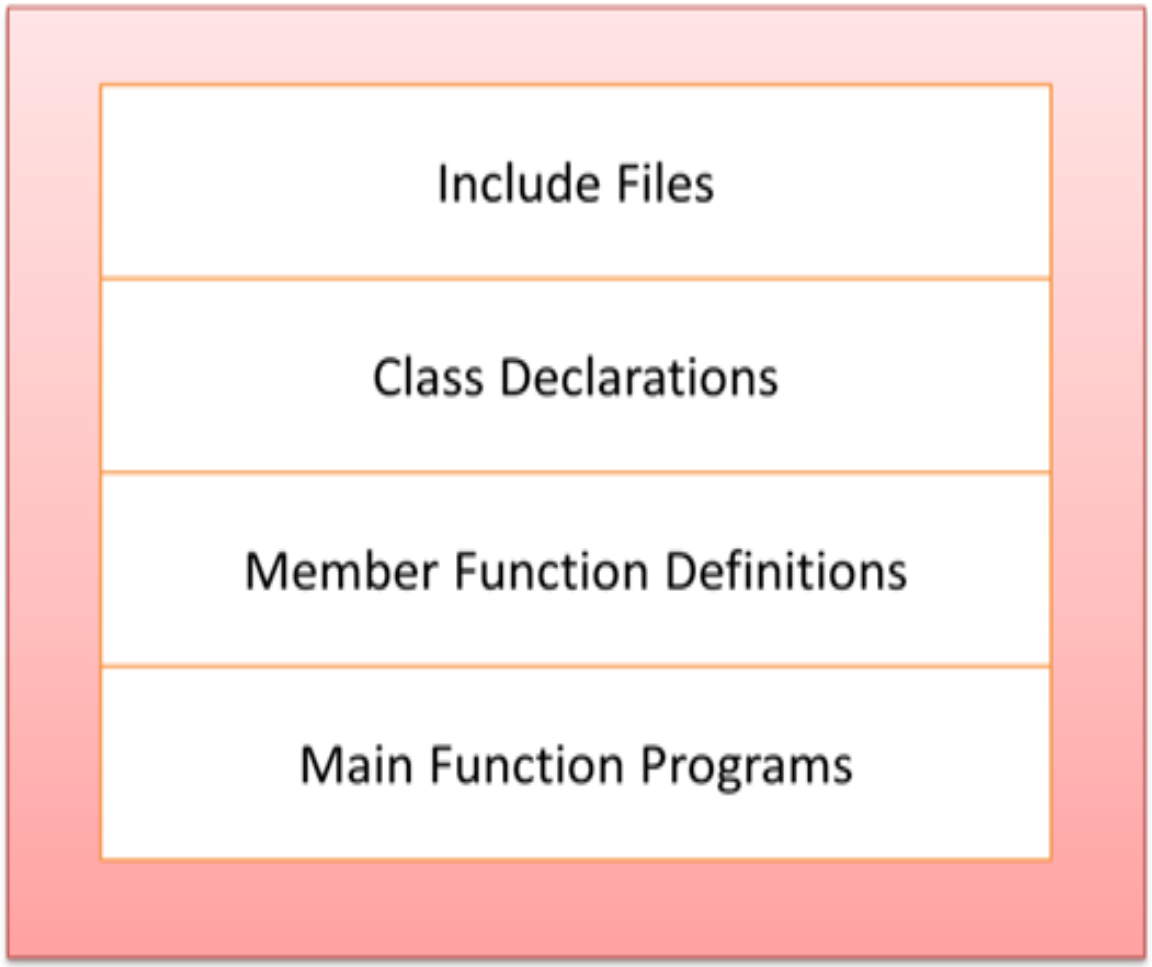
2. Object Oriented Languages

- Object Oriented Languages supports all the features of Oops including inheritance and polymorphism.
- They support built-in objects.
- C#, Java, VB. Net are the examples of object oriented languages.

C++ Concepts

- The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language.
- Object Oriented Programming is a paradigm that provides many concepts such as inheritance, data binding, polymorphism etc.
- The programming paradigm where everything is represented as an object is known as truly object-oriented programming language.
- Smalltalk is considered as the first truly object-oriented programming language.

Structure of c++ Program



Structure of c++ Program

For example,

```
#include <iostream.h>
using namespace std;
int main()
{
        cout << "Hello World!" << endl;
        return 0;
}
```

```
#include <iostream.h>
void main()
{
        float radius;
        cout << “enter radius:” ;
        cin >> radius;
        float area;       // declare area here
        area = 3.14 * radius * radius;
        cout << “area=” << area << “\n”;
        return 0;
}
```

```
#include <iostream>
using namespace std;
class Student {
        public:
                int id;              //data member (also instance variable)
                string name;  //data member(also instance variable)
        };
int main()
{
        Student s1;             //creating an object of Student
        s1.id = 201;
        s1.name = "Sonoo Jaiswal";
        cout<<s1.id<<endl;
        cout<<s1.name<<endl;
        return 0;
}
```

Comment

- C++ comment is written in one of the following two ways:
- // -- used for single line comment
- /* */ -- used for multi line comment

pre-processor directives

The pre-processors are the directives, which give instructions to the compiler to pre-process the information before actual compilation starts.

All the statements starting with the # (hash) symbol are known as pre-processor directives in C++.

There are number of preprocessor directives supported by C++ like #include, #define, #if, #else, #line, etc.

Ex. std::cin>>variable_name

Standard libraries section

- #include is a specific pre-processor command that effectively copies and pastes the entire text of the file, specified between the angle brackets, into the source code.
- The file <upstream>, which is a standard file that should come with the C++ compiler, is short for input-output streams. This command contains code for displaying and getting an input from the user.
- namespace is a prefix that is applied to all the names in a certain set. iostream file defines two names used in this program - cout and endl.
- This code is saying: Use the cout and endl tools from the std toolbox.

Main function section

- The starting point of all C++ programs is the main function.
- This function is called by the operating system when your program is executed by the computer.
- signifies the start of a block of code, and } signifies the end.

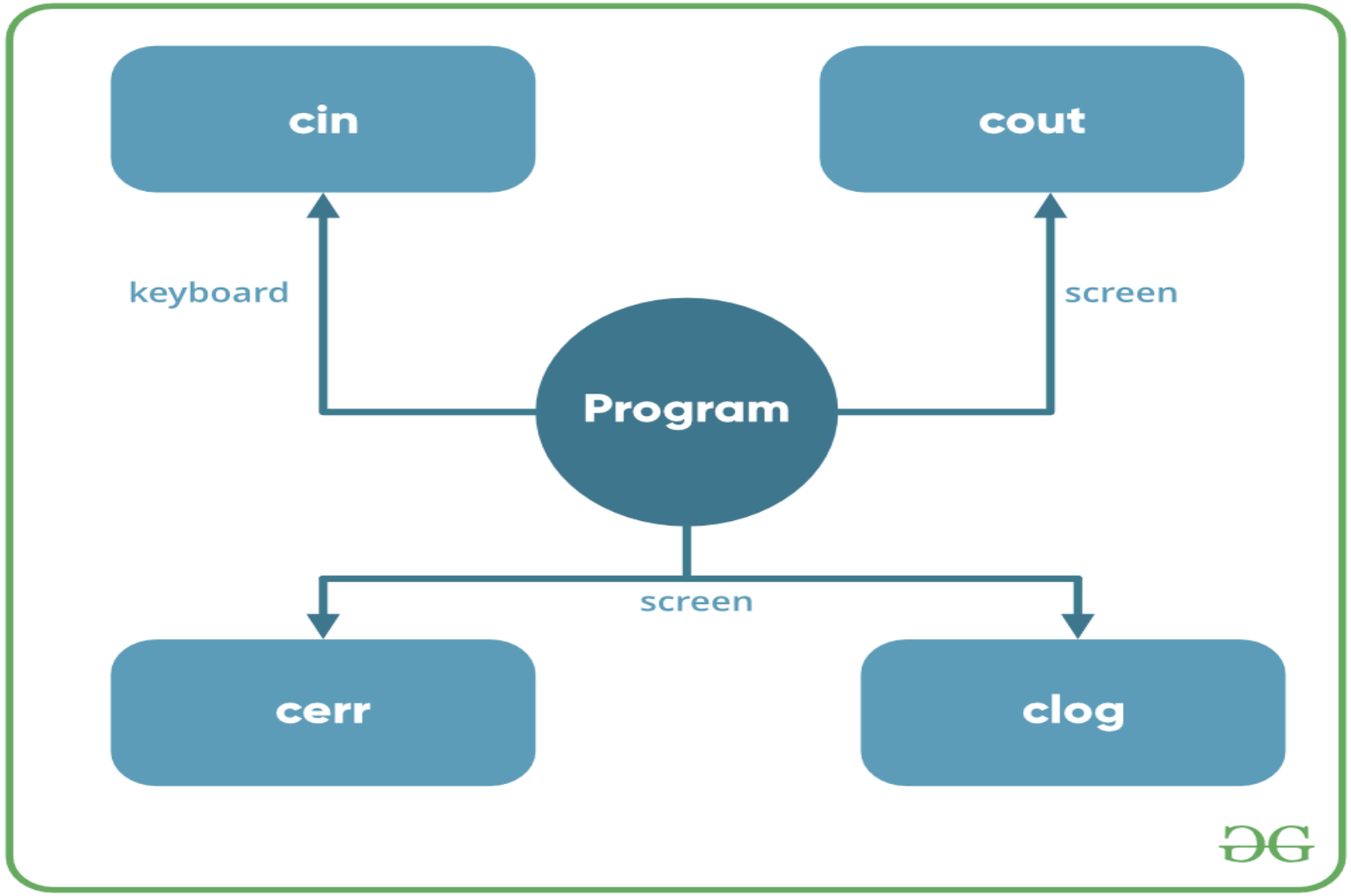
Function body section

- The name cout is short for character output and displays whatever is between the << brackets.
- Symbols such as << can also behave like functions and are used with the keyword cout.
- The return keyword tells the program to return a value to the function int main
- After the return statement, execution control returns to the operating system component that launched this program.
- Execution of the code terminates here.

Basic Input / Output in C++

- C++ comes with libraries that provide us with many ways for performing input and output.
- In C++ input and output are performed in the form of a sequence of bytes or more commonly known as streams.
- **Input Stream:** If the direction of flow of bytes is from the device(for example, Keyboard) to the main memory then this process is called input.
- **Output Stream:** If the direction of flow of bytes is opposite, i.e. from main memory to device(display screen) then this process is called output.

Basic Input / Output in C++



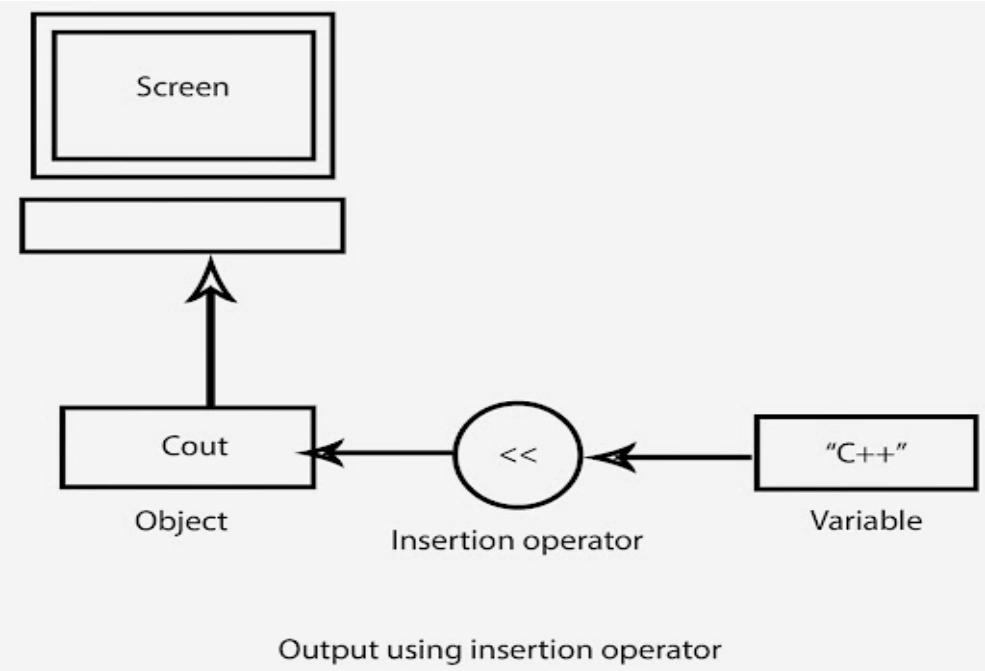
Basic Input / Output in C++

Header files available in C++ for Input/Output operations are:

- **iostream:** iostream stands for standard input-output stream. This header file contains definitions of objects like cin, cout, cerr, etc.
- **iomanip:** iomanip stands for input-output manipulators. The methods declared in these files are used for manipulating streams. This file contains definitions of setw, setprecision, etc.

The insertion operator << (Common Output - cout) :

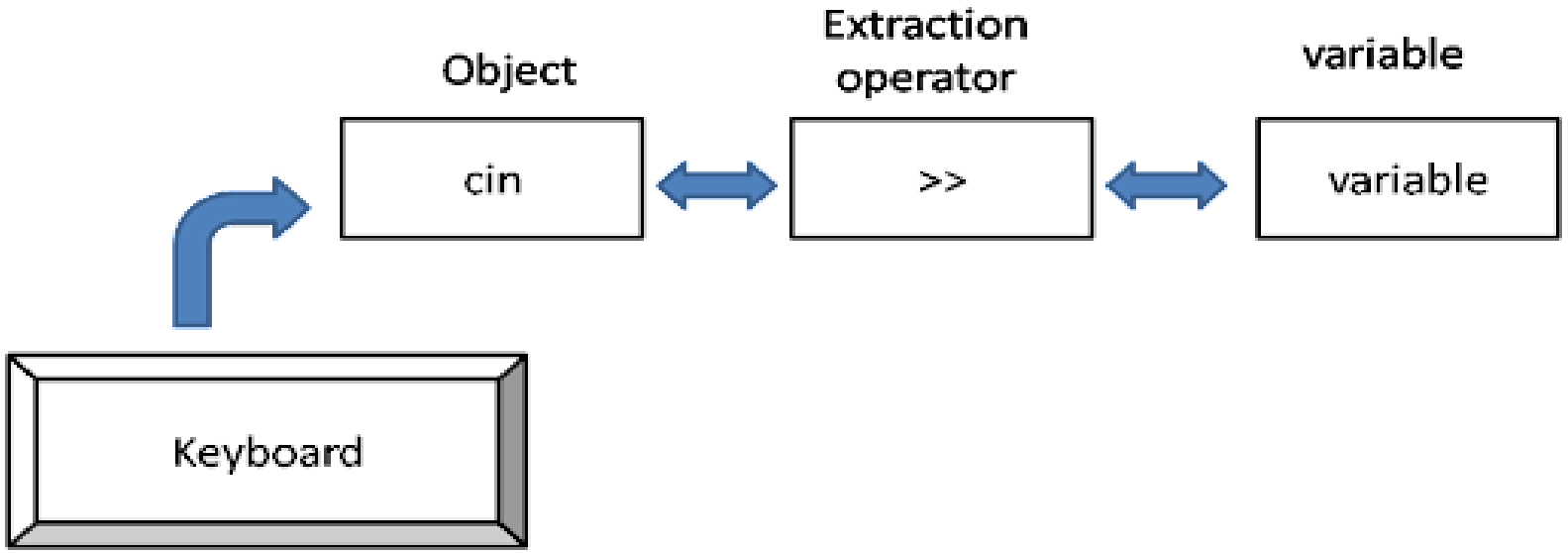
- The insertion operator (<<) points to the ostream object where in the information is inserted. cout << "hello" << "world";
- cout is an ostream object, into which information can be inserted. This stream is normally connected to the screen.It is used for displaying the message or a result of an expression.



Output using insertion operator

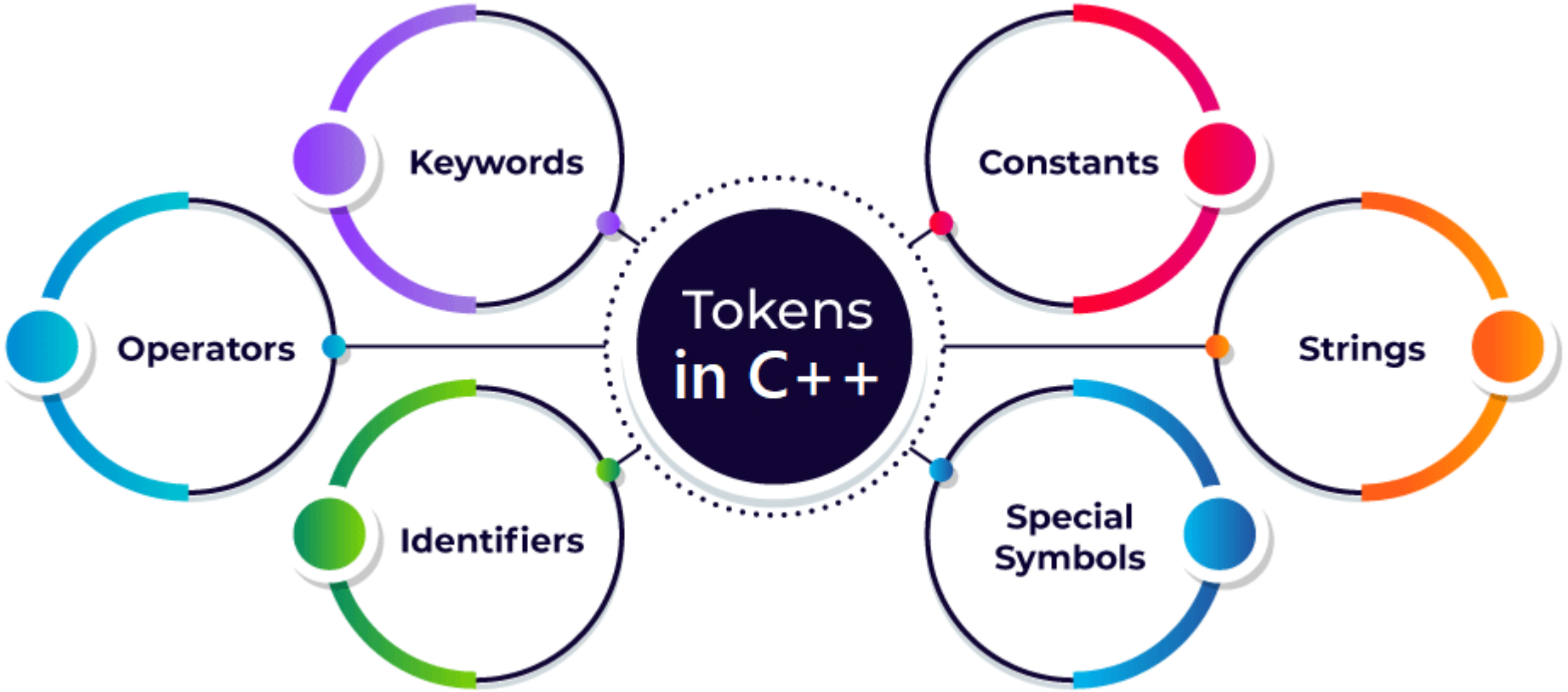
The extraction operator >> (Common Input- cin) :

- cin is an istream object from which information can be extracted. This stream is normally connected to the keyboard.
- cin is used to insert the information or the value of a variable.
- This works same as scanf() works in C language.
- Syntax : cin >> variable value;



C++ Tokens

- In C++, tokens can be defined as the smallest building block of C++ programs that the compiler understands



1. Identifiers

- In C++, entities like variables, functions, classes, or structs must be given unique names within the program so that they can be uniquely identified. The unique names given to these entities are known as identifiers.

Valid Identifiers	Invalid Identifiers
_name	#name (Cannot start with special character except '_')
Number89	2num (Cannot start with a digit)
first_name	first name (Cannot include space)
_last_name_	string (Cannot be same as a keyword)

We have to follow a set of rules to define the name of identifiers as follows:

1. An identifier can only begin with a letter or an underscore(_).
2. An identifier can consist of letters (A-Z or a-z), digits (0-9), and underscores (_). White spaces and Special characters can not be used as the name of an identifier.
3. Keywords cannot be used as an identifier because they are reserved words to do specific tasks. For example, string, int, class, struct, etc.
4. Identifier must be unique in its namespace.
5. As C++ is a case-sensitive language so identifiers such as ‘first_name’ and ‘First_name’ are different entities.

2. Keywords

- Keywords in C++ are the tokens that are the reserved words in programming languages that have their specific meaning and functionalities within a program.

break	try	catch	char	class	const	continue
default	delete	auto	else	friend	for	float
long	new	operator	private	protected	public	return
short	sizeof	static	this	typedef	enum	throw
mutable	struct	case	register	switch	and	or
namespace	static_cast	goto	not	xor	bool	do
double	int	unsigned	void	virtual	union	while

3. Constants

- Constants are the tokens in C++ that are used to define variables at the time of initialization and the assigned value cannot be changed after that.
- We can define the constants in C++ in two ways that are using the ‘const’ keyword and ‘#define’ preprocessor directive. Let’s see both methods one by one.

1. Define Constants using the ‘const’ Keyword in C++

- Constants are the values that do not change during the execution of the program once defined.
- We can make constant any type of data such as integer, float, string, and characters. A literal is a constant value assigned to a constant variable.

Syntax: const data_type variable_name = value;

2. Define Constants using the ‘#define’ preprocessor directive in C++

- #define preprocessor can be used to define constant identifiers and the identifier is replaced with the defined value throughout the program where ever it is used.
- It is defined globally outside the main function.

Syntax:

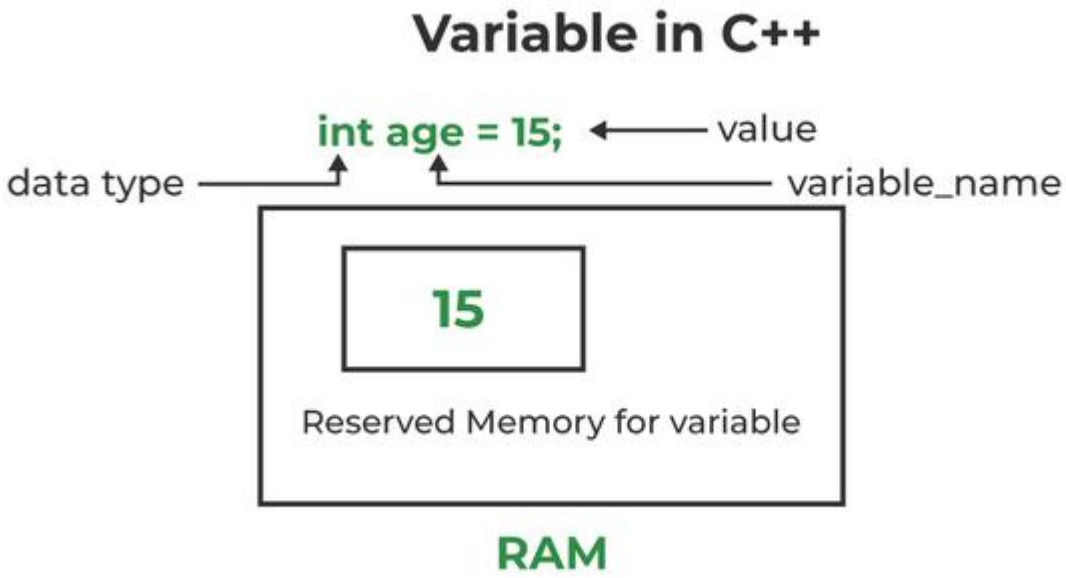
#define constant_Name value

#define PI 3.14

```
#include<iostream.h>
#define PI 3.14
void main()
{
Float radius, area;
//const  float PI=3.14;
cout<<"Enter the radius of circle: ";
cin>>radius;
area=PI*radius*radius;
cout<<"Area of circle is: "<<area;
getch();
}
```

Variable

- Variables in C++ is a name given to a memory location. It is the basic unit of storage in a program.
- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location.
- In C++, all the variables must be declared before use.



Reference Variable

- Reference variable is an alternate name of already existing variable. It cannot be changed to refer another variable and should be initialized at the time of declaration and cannot be NULL.
- The operator '&' is used to declare reference variable.

Syntax of reference variable

```
datatype variable_name;                    // variable declaration
datatype& refer_var = variable_name; // reference variable
```

Function Parameters

- References can also be passed as a function parameter. It does not create a copy of the argument and behaves as an alias for a parameter.

Example

```
#include <iostream>

using namespace std;

int main() {
   int a = 8;
   int& b = a;
   cout << "The variable a : " << a;
   cout << "\nThe reference variable r : " << b;
   return 0;
}
```

```
#include <iostream>
Void main()  {
        int a=9,b=10;
        swap(a, b);
        cout << "value of a is :" <<a<< std::endl;
        cout << "value of b is :" <<b<< std::endl;
}
void swap(int p, int &q){
        int temp;
        temp=p;
        p=q;
        q=temp;
}
```

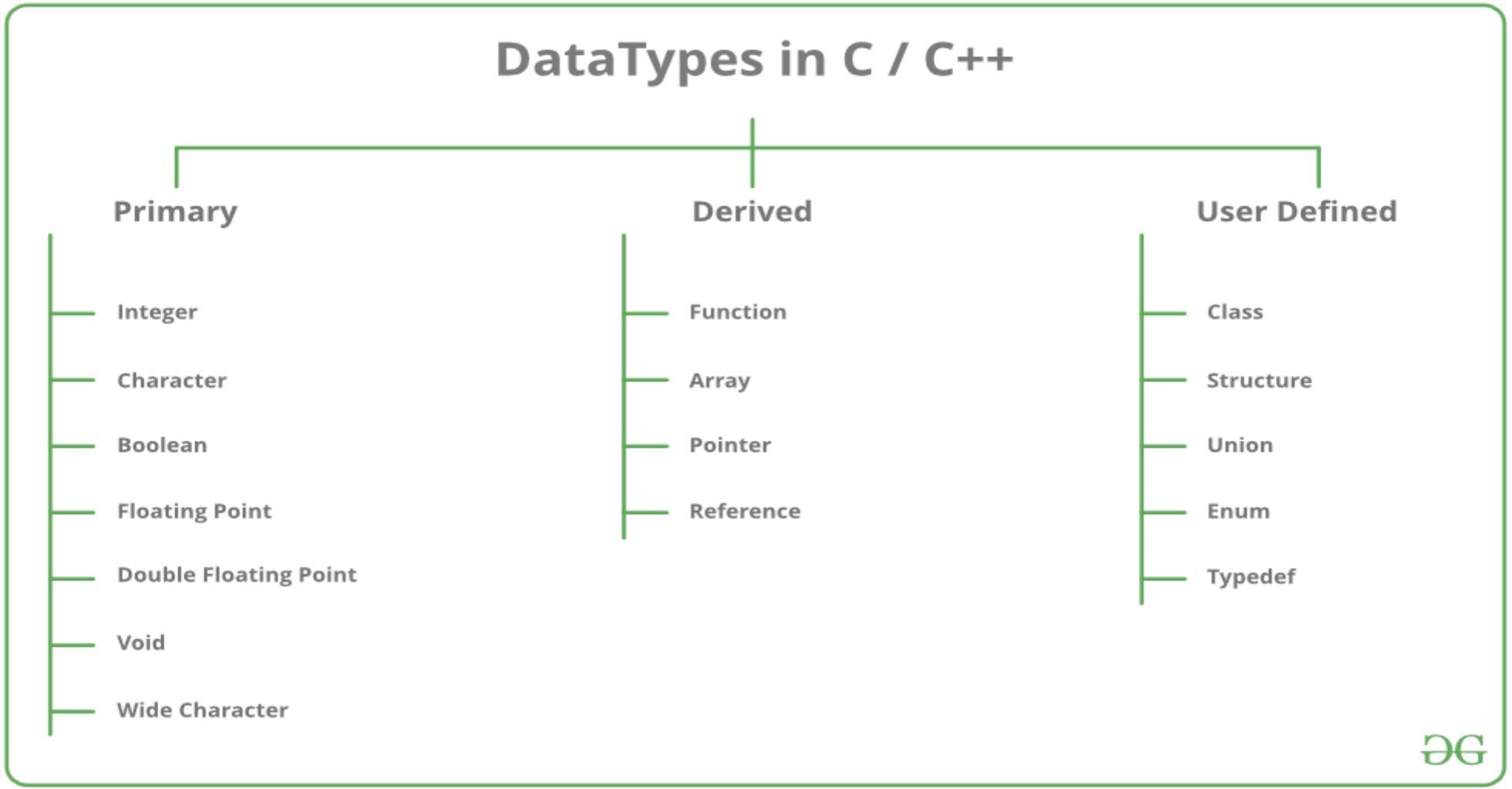
Output:

value of a is :10

value of b is :9

Basic Data types

- Data Types in C++ are divided into 3 Types:



Data Type	Meaning	Size (in Bytes)
int	Integer	2 or 4
float	Floating-point	4
double	Double Floating-point	8
char	Character	1
wchar_t	Wide Character	2
bool	Boolean	1
void	Empty	0

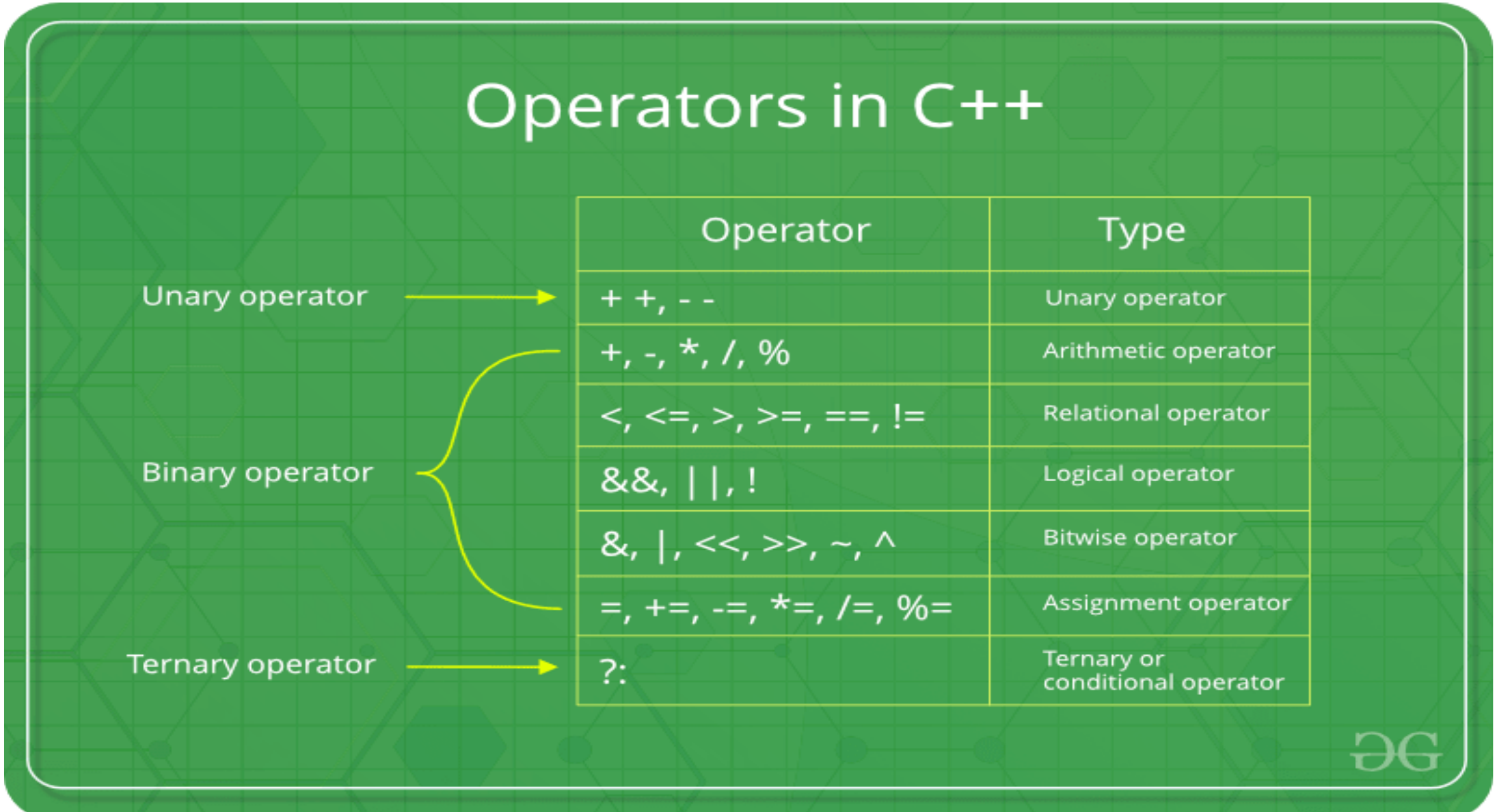
Operators in C++

- An operator is a symbol that operates on a value to perform specific mathematical or logical computations.

Operators in C++ can be classified into 6 types:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Ternary or Conditional Operators

Operators in C++



Special operators in c++

New Operators

Operators	Description
::	Scope resolution operator
::*	Pointer-to-member declarator
->*	Pointer-to-member operator
.*	Pointer-to-member operator
new	Memory allocation operator
endl	Line feed operator
setw	Field width operator
delete	Memory release operator

Scope Resolution Operators (::)

- **Local variable:** A variable which is declared inside the function or block is called as the local variable for that function and block.
- **Global variable**: A variable which is declared outside the function or block is called as the global variable for that function and block.
- In C, to access the global version of the variable inside the function and block that is not possible.
- In C++, the global version of variable is accessed inside the function and block using the scope resolution operator.
- Scope resolution operator is used to uncover a hidden variable.

Syntax: **:: variable-name**

Example:

```
#include <iostream>

int a = 10;

void main( )

{

        int a = 15;

        cout<< “ Value of a = ” << a<<endl;

        cout<< “ Value of a = ” << ::a;

}
```

Output:

Value of a = 15 Value of a = 10

Pointer to Member Operators

S.N.	Operator	Meaning
1	::*	Pointer-to-Member Declarator. It is used to declare a pointer to a member of a class.
2	->*	Pointer-to-Member. Used to access a member using a pointer to an object and a pointer to that member.
3	.*	Pointer-to-Member. Used to access a member using an object and a pointer to that member.

Memory Management Operators

- In C, calloc( ) and malloc( ) function are used to allocate memory dynamically at runtime. Similarly, it uses the function free( ) to free dynamically allocated memory.
- In C++, the unary operators new and delete are used to allocate and free a memory at run-time.

new operator

- An object can be created by using new and destroyed by using delete operator.
- The new operator can be used to create objects of any type.

Syntax: pointer-variable = new data-type;

Example: int *p = new int;

float *q = new float;

delete operator

- When a data object is no longer needed, it is destroyed to release the memory space for reuse.

Syntax:

delete pointer-variable;

Example:

delete p; delete q;

Example:

```
#include <iostream>
Void main()
{
    int* p = new int;
   *p = 5;
   std::cout << *p;
   delete p;
}
```

Operators Precedence:

- Operator precedence determines the grouping of terms in an expression.
- The associativity of an operator is a property that determines how operators of the same precedence are grouped in the absence of parentheses.
- Certain operators have higher precedence than others; **for example**, the multiplication operator has higher precedence than the addition operator:
- **For example** x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Manipulators

- **Manipulators** are helping functions that can modify the input/output stream.
- It does not mean that we change the value of a variable, it only modifies the I/O stream using insertion (<<) and extraction (>>) operators.
- To access manipulators, the file iomanip.h should be included in the program.
- Manipulators are operators that are used to format the output.
- Most commonly used manipulators are endl and setw.

Manipulators

- The **endl** manipulator is used in the output statement when we want to insert a linefeed. It has the same effect as using the newline character "\n".
- **Setw:** Setw function sets the field width or number of characters that are to be displayed before a particular field.

  Syntax: setw(width) Where, width specifies the field width.
- **Setprecision:** setprecision manipulator sets the total number of digits to be displayed, when floating point numbers are printed.
- **Setfill:** Setfill function is used to fill the stream with char type c specified as a parameter.

Example:

```
#include<iostream.h>
#include<iomanip.h>
void main( )
{
        int a=15,b=420,c=3542;
        cout<< “a =”<<setw(4)<<a<<endl;
        cout<< “b =”<<setw(4)<<b<<endl;
        cout<< “c =”< setw(4)<<c<<endl;
}
```

Output:

```
a =    15
b = 420
c =3542
```

Example:

```
#include<iostream.h>
#include<iomanip.h>
void main( )
{
        cout<< setw(10) <<11<<"\n";
        cout<< setw(10) <<2222<<"\n";

        cout<< setfill('0');
        cout<< setw(10) <<11<<"\n";
        cout<< setw(10) <<2222<<"\n";

        cout<< setfill('-')<< setw(10) <<11<<"\n";
        cout<< setfill('*')<< setw(10) <<2222<<"\n";
          return 0;
}
```

Manipulators

- **Setprecision:** setprecision manipulator sets the total number of digits to be displayed, when floating point numbers are printed.
- The C++ setprecision can also be used to format only the decimal places instead of the whole floating-point or double value.
- This can be done using the fixed keyword before the setprecision() method.
- When the fixed keyword is used, the argument in the setprecision() function specifies the number of decimal places to be printed in the output.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main ()
{
        double float_value =3.14159;
        cout << setprecision(4) << float_value << '\n';
        cout << setprecision(5) << float_value << '\n';
        cout << setprecision(9) << float_value << '\n';
        cout << fixed;
        cout << setprecision(5) << float_value << '\n';
        cout << setprecision(10) << float_value << '\n';
        return 0;
}
```

Output:

```
3.142
3.1416
3.14159
3.14159
3.1415900000
```

Enumeration

- An enumeration is a user-defined data type that consists of integral constants.
- To define an enumeration, keyword enum is used.
- It can be assigned some limited values.
- These values are defined by the programmer at the time of declaring the enumerated type.

Syntax:

enum enumerated-type-name { value1, value2, value3…..valueN };

Example:

```
enum week { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };
int main()
{
        week today;
        today = Wednesday;
        cout << "Day " << today+1;
        return 0;
}
```

Output:

Day 4

Example:

```
#include <iostream>
enum seasons { spring = 34, summer = 4, autumn = 9, winter = 32};
int main()
{
    seasons s;
    s = summer;
    cout << "Summer = " << s << endl;
    return 0;
}
```

Output: